CS 3 Final Review

Gilbert Chou, Jenny Franco
and Colleen Lewis

December 14, 2008
1-4pm GPB

Warm-up!

What is your favorite color?

Option A: `Brown`

Option B: `Orange`

Option C: `Yellow`

Option D: `Green`

Predict the Output

Predict the output

`((repeated bf 3) '(cat dog hat bat))` ➔ _____

Option A: `Error`

Option B: `'bat`

Option C: `bat`

Option D: `(bat)`

Answer Predict the output

`((repeated bf 3) '(cat dog hat bat))` ➔ _____

Option A: `Error`

Option B: `'bat`

Option C: `bat`

Option D: `(bat)`

- `butfirst` of a sentence ALWAYS returns a sentence
- Never quote the output of scheme!
- `(repeated bf 3)` returns a procedure that takes a word or a sentence.

Predict the output

`(cons '() (list '() (cons '() '()))) ➔ _____`

Option A: `Error`

Option B: `( () () (()) )`

Option C: `( () () () () )`

Option D: `( () () (() ()) )`

Answer Predict the output

`(cons '() (list '() (cons '() '()))) ➔ _____`

Option A: `Error`

Option B: `( () () (()) )`

Option C: `( () () () () )`

Option D: `( () () (() ()) )`

- Review what cons, append and list do…

Predict the output

`(empty? '()) ➔ _____`

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `no idea!`

Answer Predict the output

`(empty? '()) ➔ _____`

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `no idea!`

- `empty?` returns true for empty sentences and words
- Don't use it with lists!

Predict the output

`(empty? "") ➔ _____`

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `no idea!`

Answer Predict the output

`(empty? "") ➔ _____`

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `no idea!`

- `empty?` returns true for empty sentences and words
- Don't use it with lists!

Predict the output

`(null? "") ➔ _____`

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `no idea!`

Answer Predict the output

`(null? “”) ➔ _____`

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `no idea!`

- `null?` returns true for null lists
- Don’t use it with words and sentences!

Predict the output

`(null? (cons “” ‘())) ➔ _____`

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `no idea!`

Answer Predict the output

`(null? (cons “” ‘())) ➔ _____`

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `no idea!`

- An empty word counts as something inside the list. So – it is not `null?`

Predict the output

`(assoc ‘x ‘((a 1) (b 2) (x 3) (x 4)) ➔ _____`

- Option A: `Error`
- Option B: `3`
- Option C: `(x 3)`
- Option D: `(x 4)`

Answer Predict the output

`(assoc ‘x ‘((a 1) (b 2) (x 3) (x 4)) ➔ _____`

- Option A: `Error`
- Option B: `3`
- Option C: `(x 3)`
- Option D: `(x 4)`

- `assoc` just returns the first match.
- Returns the entire pair – not just the `car` of the `cdr`

Predict the output

`(member ‘x ‘(a b c d x e f g)) ➔ _____`

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `something else`

Answer Predict the output

`(member 'x '(a b c d x e f g))` ➔ _____

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `something else`

- Returns: `(x e f g)`
- `member` returns the rest of the sentence starting with the thing you were looking for. It is a **semi-predicate**
- `member?` returns just #t or #f

Predict the output

`(member? 'x '(a b c d x e f g))` ➔ _____

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `something else`

Answer Predict the output

`(member? 'x '(a b c d x e f g))` ➔ _____

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `something else`

- `member` returns the rest of the sentence starting with the thing you were looking for. It is a **semi-predicate**
- `member?` returns just #t or #f

Predict the output

`(member? 'x '(a b c d (x) e f g))` ➔ _____

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `something else`

Answer Predict the output

`(member? 'x '(a b c d (x) e f g))` ➔ _____

- Option A: `Error`
- Option B: `#t ;; True`
- Option C: `#f ;; False`
- Option D: `something else`

- `member` doesn't look inside of sublists

Predict the output

`(cons 1 2)` ➔ _____

- Option A: `Error`
- Option B: `(1 2)`
- Option C: `(1 . 2)`
- Option D: `something else`

Answer Predict the output

`(cons 1 2)` ➔ _____

- Option A: `Error`
- Option B: `(1 2)`
- Option C: `(1 . 2)`
- Option D: `something else`

- The second argument to `cons` should be a list otherwise you end up with this weird dot.
- Remember cons pulls out the "underwear" of the second argument and drops the first argument in.

Predict the output

`(map cons 1 '(2))` ➔ _____

- Option A: `Error`
- Option B: `(1 2)`
- Option C: `(1 . 2)`
- Option D: `((1 2))`

Answer Predict the output

`(map cons 1 '(2))` ➔ _____

- Option A: `Error`
- Option B: `(1 2)`
- Option C: `(1 . 2)`
- Option D: `((1 2))`

- Map takes in a procedure and one or more LISTS!
- There should be the same number of lists as there are arguments that the procedure takes in!

Predict the output

`(map cons '(1) '(2))` ➔ _____

- Option A: `Error`
- Option B: `(1 2)`
- Option C: `(1 . 2)`
- Option D: `((1 2))`

Answer Predict the output

`(map cons '(1) '(2))` ➔ _____

- Option A: `Error`
- Option B: `(1 2)`
- Option C: `(1 . 2)`
- Option D: `((1 . 2))`

- Map takes the car of each list and calls the procedure `cons` with them. i.e. `(cons 1 2)` it does that for each element in the lists and puts the result in a list!

Fill in the blank

_____ ➔ `((1 2) (A B))`

- Option A: `(map cons '(1 A) '((2) (B)))`
- Option B: `(map cons '((1) (A)) '((2) (B)))`
- Option C: `(map cons '(1 A) '(2 B))`
- Option D: `something else`

Answer Fill in the blank

_____ ➔ `'((1 2) (A B))`

Option A: `(map cons '(1 A) '((2) (B)))`
- `(list (cons 1 '(2)) (cons A '(B)))`

Option B: `(map cons '((1) (A)) '((2) (B)))`
- `(list (cons '(1) '(2)) (cons '(A) '(B)))`

Option C: `(map cons '(1 A) '(2 B))`
- `(list (cons 1 2) (cons A B))`

Option D: `something else`

Predict the output

`(='x 'x)` ➔ _____

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `something else`

Answer Predict the output

`(='x 'x)` ➔ _____

- `=` only works with numbers! For words/sentences use `equal?`

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `something else`

Predict the output

`(='1 1)` ➔ _____

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `something else`

Answer Predict the output

`(='1 1)` ➔ _____

- Numbers are words!
- The quote is implicit on numbers

Option A: `Error`

Option B: `#t ;; True`

Option C: `#f ;; False`

Option D: `something else`

Predict the output

`(every first 'awesome)` ➔ _____

Option A: `Error`

Option B: `a`

Option C: `awesome`

Option D: `(a w e s o m e)`

Answer

Predict the output

(every first 'awesome) ➔ _____

Option A: Error

Option B: a

Option C: awesome

Option D: (a w e s o m e)

- Every returns a sentence even if it is given a word!

```
1  (define (depth lst)
2    (if (not (list? lst))
3        0
4       (max (map (+ depth 1) lst))))
```

Option A: Error

Option B: 0

Option C: 1

Option D: something else

(depth 1234567)➔ _____

Answer

```
1  (define (depth lst)
2    (if (not (list? lst))
3        0
4       (max (map (+ depth 1) lst))))
```

Option A: Error

Option B: 0

Option C: 1

Option D: something else

(depth 1234567)➔ _____

```
1  (define (depth lst)
2    (if (not (list? lst))
3        0
4       (max (map (+ depth 1) lst))))
```

Option A: Error

Option B: 0

Option C: 1

Option D: something else

(depth '((a)(b)))➔ _____

Answer

```
1  (define (depth lst)
2    (if (not (list? lst))
3        0
4       (max (map (+ depth 1) lst))))
```

Option A: Error

Option B: 0

Option C: 1

Option D: something else

(depth '((a)(b)))➔ _____

```
1  (define (depth lst)
2    (if (not (list? lst))
3        0
4       (max (map (+ depth 1) lst))))
```

Option A: 1

Option B: 2

Option C: 3

Option D: 4

Which line has an error? _____

Answer

```
1 (define (depth lst)
2   (if (not (list? lst))
3       0
4     (max (map (+ depth 1) lst))))
```

Option A: 1

Option B: 2

Option C: 3

Option D: 4

Which line has an error? _____

```
1 (define (depth lst)
2   (if (not (list? lst))
3       0
4     (max (map (+ depth 1) lst))))
```

Correct the error!

_____

Answer

```
1 (define (depth lst)
2   (if (not (list? lst))
3       0
4     (max (map (+ depth 1) lst))))
```

New line 4:

```
(+ 1 (reduce max (map depth lst)))
```

List or Sentence?

Would you use this with Lists or Sentences?

`item`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

Answer

Would you use this with Lists or Sentences?

`item`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

- `list-ref` for lists
- `item` for sentences/words

Would you use this
with Lists or Sentences?

`assoc`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

Answer Would you use this
with Lists or Sentences?

`assoc`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

- Assoc lists are LISTS

Would you use this
with Lists or Sentences?

`first`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

Answer Would you use this
with Lists or Sentences?

`first`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

- `car` for lists
- `first` for sentences/words

Would you use this
with Lists or Sentences?

`reduce`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

Answer Would you use this
with Lists or Sentences?

`reduce`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

- `reduce` for lists
- `accumulate` for sentences/words

Would you use this
with Lists or Sentences?

`map`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

Answer

Would you use this
with Lists or Sentences?

`map`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

- `map` for lists (and it is way cooler than `every`!)
- `every` for sentences/words

Would you use this
with Lists or Sentences?

`filter`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

Answer

Would you use this
with Lists or Sentences?

`filter`

Option A: `lists`

Option B: `sentences`

Option C: `neither`

Option D: `both`

- `filter` for lists
- `keep` for sentences/words